Creating a simple and extendable web based course framework with Flash CS3

By Matt Perkins, kheavy@nudoru.com

Last updated: 5.20.08

This example will walk though the creation of a simple web based training (WBT) framework in Flash CS3 and Actionscript 3. You should be familiar with the basic concepts of programming in AS3 including: creating functions, using variables, using the display list and how to import and use classes. This isn't a tutorial, but shows the code and the steps involved.

This example makes use of the Pipwerks Flash/SCORM API for Flash and AS3 (available here http://pipwerks.com/lab/downloads.php), SWFObject and two classes of mine: one for simplifying LMS communication and one for tracking time.

There are many different ways to create a course framework in Flash, but this is the method that I've been using for the last four years. I find the timeline based approach has limitations once you start editing or maintaining the course. This method makes adding, removing pages and editing pages simple since it's all dynamic and the pages are isolated in their own library objects.

The framework that I'll create is pretty simple from a content structure perspective. There is no page grouping, like lessons, but that isn't a limitation since it allows for the creation of a perfectly fine SCO.

Note: *This code has been tested in Bank of America's Saba LMS and verified as working.*

Step 1 – Set up the files

`Root folder`

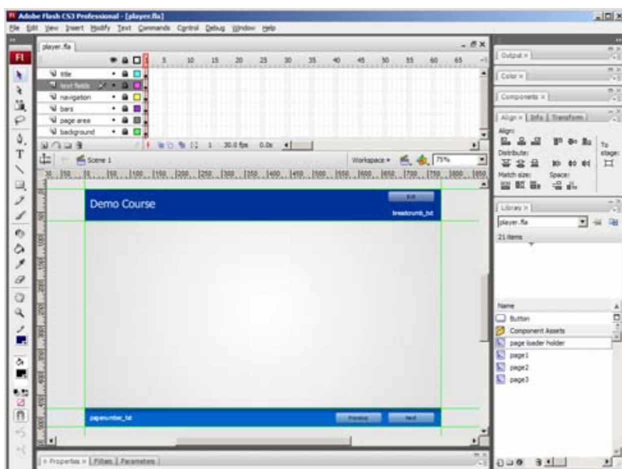
- `\classes`
    - `\com`
    - `\pipwerks`
    - `LMSData.as`
- `\embed`
- `player.fla`
- `Player.as`
- `player.swf`
- `index.html`
- `SCORM_API_wrapper_v1.1.5.js`
- `imsmanifest.xml`

The file and folder structure for this example is shown to the left. The classes that we'll use are in the "classes" folder. The files for SWFObject (used for embedding the player SWF in the HTML page) are in the "embed" folder. The source code that this example covers is in the root folder.

Note: *make sure that the "class path" of the FLA file points to the "classes" folder. Flash won't be able to find the necessary classes if it's not.*

Step 2 – Create the user interface

I created user interface in the Flash 9 authoring environment. It includes the basics for a simple WBT – a title, breadcrumb bar (for showing the current page title), a page number, and navigation buttons.



The file is organized into layers representing the different visual areas:

- Title
- Text fields – bread crumb and page number
- Navigation
- "Bars" – the blue behind the text and navigation
- Page layer – contains an instance of a transparent sprite from the library where we'll add the pages to show them
- Background

When pages display in the WBT, they will be added to the “page area” sprite. I did it this way so that they will properly layer with the other interface elements. This is an easy way to manage layers in the user interface. If I had chosen to add them to the stage at the highest layer, they could potentially overlap other areas of the screen and cover the navigation buttons or other important interface elements.

Step 3 – Creating the document class

This is a very basic document class for an object oriented Flash AS3 project. It doesn’t do much more than import a few key classes and trace a message to the output window.

The name for this file is “Player.as” and it’s saved in the root folder with “player.fla.”

Note: *in the examples, new code will appear in brown.*

```
package {

	// import the classes so that we can work with the items on the screen
	import flash.display.*;
	import flash.text.*;

	// a document class must expent either Sprite or Movieclip
	// since all of our code is in the document class and we have no timeline animation, we'll extend the Sprite class
	public class  Player extends Sprite	{

		public function Player():void {
			trace("Hello!");
		}

	}

}
```

Step 4 – Creating the page data structure

I've added an array called "_PageList" that will hold the pages in the course. To hold the information for each page, I'm using an "object" datatype for simplicity's sake. An object is a data type that can have dynamic properties – in our case: page title and linkage ID. But as a future enhancement, these could easily be expanded upon later.

To make it easier to add pages to the array, I created a function "addNewPage()" that takes the title and linkage ID as parameters, creates the object and pushes to the page list array.

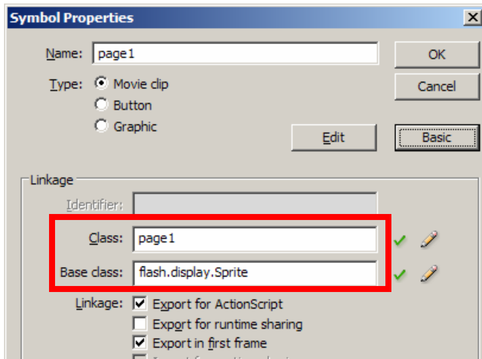
Since we're using an array, we need a way to keep track of the current page. The index of where we are in the array is the most direct way.

```
package {

        // import the classes so that we can work with the items on the screen
        import flash.display.*;
        import flash.text.*;

        // a document class must expent either Sprite or Movieclip
        // since all of our code is in the document class and we have no timeline animation, we'll extend the Sprite class
        public class  Player extends Sprite      {

                // create an array to hold the information for the pages
                var _PageList                   :Array;
                // index of the current page
                var _CurrentPageIdx:     int;

                public function Player():void {
                        // initialize the page list array
                        _PageList = new Array();
                        // set the index to 0 - the first page
                        _CurrentPageIdx = 0;

                        // setup the pages
                        addNewPage("Course Introduction", "page1");
                        addNewPage("The Content", "page2");
                        addNewPage("Conclusion", "page3");

                        trace("Ready to go ...");
                }

                // will add a new page object to the page list
                // title is the name of the page, linkage is the ID of the page sprite in the library
                private function addNewPage(title:String, linkage:String):void {
                        // create a new object to hold details of the page
                        var pObject:Object = new Object();
                        // assign the title and linkage to properties of the object
                        pObject.title = title;
                        pObject.linkage = linkage;
                        // add it to the page list
                        _PageList.push(pObject);
                        trace("Added page '"+title+"' to the course");
                }
        }
}
```

Step 5 – Displaying a page

Displaying a page is pretty straight forward: 1) get the linkage ID of the current page and 2) add it to the display list. The trickiest part is that in AS3 you need to specify a class name for the library object, a string doesn't work like it did in AS2. So to do this you need to: 1) get a reference to the class using "getDefinitionByName" and 2) create a new instance of it making sure to cast it as a Sprite data type.

To maintain good layering, I'm adding the pages, not directly to the stage, but to a sprite that I placed on the stage in the authoring tool. This is so that I can control where the sprite is layered in the interface and keep it from showing any content that's layered on top of any other interface element.



Note: *AS3 introduces a new type of display object: Sprite. Sprites don't have a time line and can't have any code in them. And they take up less memory and use fewer resources than MovieClips, so it's best to use them when you can.*

In the screen shot to the left, the "Class" is the "linkage ID" and the "Base class" sets the library symbol to extend "Sprite." All of the pages in the example are setup this way.

```
package {

	// import the classes so that we can work with the items on the screen
	import flash.display.*;
	import flash.text.*;
	// utility class that let's us get references to the page Sprites in the library
	import flash.utils.getDefinitionByName;

	// a document class must expent either Sprite or Movieclip
	// since all of our code is in the document class and we have no timeline animation, we'll extend the Sprite class
	public class  Player extends Sprite 	{

		// create an array to hold the information for the pages
		var _PageList			:Array;
		// index of the current page
		var _CurrentPageIdx:	int;
		// reference to the "page layer" in the user interface
		var _PageDisplayTarget	:Sprite;
		// the sprite of the currently displaying page
		var _CurrentPageSprite	:Sprite;

		public function Player():void {
			// initialize the page list array
			_PageList = new Array();
			// set the index to 0 - the first page
			_CurrentPageIdx = 0;
			// set the page layer target
			_PageDisplayTarget = pagelayer_mc;

			// setup the pages
			addNewPage("Course Introduction", "page1");
			addNewPage("The Content", "page2");
			addNewPage("Conclusion", "page3");

			trace("Ready to go ...");
```

```
                // show the current page
                displayCurrentPage();
        }

        // show the current page in the interface
        private function displayCurrentPage():void {
                // is there a page currently showing? remove it
                if (_CurrentPageSprite) _PageDisplayTarget.removeChild(_CurrentPageSprite);
                // get a reference to the Sprite in the library
                var _ClassRef:Class = getDefinitionByName(_PageList[_CurrentPageIdx].linkage) as Class;
                _CurrentPageSprite =  new _ClassRef() as Sprite;
                // show the page - add it to the display list of the page layer
                _PageDisplayTarget.addChild(_CurrentPageSprite);
        }

        // will add a new page object to the page list
        // title is the name of the page, linkage is the ID of the page sprite in the library
        private function addNewPage(title:String, linkage:String):void {
                // create a new object to hold details of the page
                var pObject:Object = new Object();
                // assign the title and linkage to properties of the object
                pObject.title = title;
                pObject.linkage = linkage;
                // add it to the page list
                _PageList.push(pObject);
                trace("Added page '"+title+"' to the course");
        }
    }
}
```

Step 6 – Updating the interface based on the current page

Here, I'm updating the two text fields' content based on the current page and where that page is in relation to the other pages in the course.

I'm also creating two support functions to determine if there is a page to either side of the current one. They are used to enable or disable the previous and next navigation buttons and later on for navigating in the course.

```
package {

	// import the classes so that we can work with the items on the screen
	import flash.display.*;
	import flash.text.*;
	// utility class that let's us get references to the page Sprites in the library
	import flash.utils.getDefinitionByName;

	// a document class must expent either Sprite or Movieclip
	// since all of our code is in the document class and we have no timeline animation, we'll extend the Sprite class
	public class  Player extends Sprite	{

		// create an array to hold the information for the pages
		var _PageList			:Array;
		// index of the current page
		var _CurrentPageIdx:	int;
		// reference to the "page layer" in the user interface
		var _PageDisplayTarget	:Sprite;
		// the sprite of the currently displaying page
		var _CurrentPageSprite	:Sprite;

		// getter function to get the number of pages
		public function get numPages():int { return _PageList.length }

		public function Player():void {
			// initialize the page list array
			_PageList = new Array();
			// set the index to 0 - the first page
			_CurrentPageIdx = 0;
			// set the page layer target
			_PageDisplayTarget = pagelayer_mc;

			// setup the pages
			addNewPage("Course Introduction", "page1");
			addNewPage("The Content", "page2");
			addNewPage("Conclusion", "page3");

			trace("Ready to go ...");

			// show the current page
			displayCurrentPage();
		}

		// show the current page in the interface
		private function displayCurrentPage():void {
			// is there a page currently showing? remove it
			if (_CurrentPageSprite) _PageDisplayTarget.removeChild(_CurrentPageSprite);
			// get a reference to the Sprite in the library
			var _ClassRef:Class = getDefinitionByName(_PageList[_CurrentPageIdx].linkage) as Class;
			_CurrentPageSprite =  new _ClassRef() as Sprite;
			// show the page - add it to the display list of the page layer
			_PageDisplayTarget.addChild(_CurrentPageSprite);
			// update the interface based on the current page
			updateInterface();
		}
```

```
		// updates the text fields and buttons based on the current page
		private function updateInterface():void {
			// have to add 1 to the current page index variable since array index start at 0
			pagenumber_txt.text = "Page " + (_CurrentPageIdx + 1) + " of " + numPages;
			// update the bread crumb with the page title
			breadcrumb_txt.text = _PageList[_CurrentPageIdx].title;
			// turn the navigation on or off based on where the learner is
			if (isNextPage()) next_btn.enabled = true;
				else next_btn.enabled = false;
			if (isPreviousPage()) prev_btn.enabled = true;
				else prev_btn.enabled = false;
		}

		// is there a next page from the current page?
		private function isNextPage():Boolean {
			// if the current page index greater than or the total number of pages?
			if (_CurrentPageIdx >= numPages - 1) return false;
			return true;
		}

		// is there a previous page from the current page?
		private function isPreviousPage():Boolean {
			// is the learner on the first page?
			if (_CurrentPageIdx == 0) return false;
			return true;
		}

		// will add a new page object to the page list
		// title is the name of the page, linkage is the ID of the page sprite in the library
		private function addNewPage(title:String, linkage:String):void {
			// create a new object to hold details of the page
			var pObject:Object = new Object();
			// assign the title and linkage to properties of the object
			pObject.title = title;
			pObject.linkage = linkage;
			// add it to the page list
			_PageList.push(pObject);
			trace("Added page '"+title+"' to the course");
		}
	}
}
```

Step 7 – Navigating between pages

Now the code will respond when the learner clicks the navigation buttons. The next and previous navigation functions simply check to see if there is a page in the direction we want to go, and if so, add 1 or subtract 1 from the current index and update the display.

```
package {

        // import the classes so that we can work with the items on the screen
        import flash.display.*;
        import flash.text.*;
        // so that it can respond to events
        import flash.events.*;
        // utility class that let's us get references to the page Sprites in the library
        import flash.utils.getDefinitionByName;

        // a document class must expent either Sprite or Movieclip
        // since all of our code is in the document class and we have no timeline animation, we'll extend the Sprite class
        public class  Player extends Sprite      {

                // create an array to hold the information for the pages
                var _PageList                   :Array;
                // index of the current page
                var _CurrentPageIdx:     int;
                // reference to the "page layer" in the user interface
                var _PageDisplayTarget   :Sprite;
                // the sprite of the currently displaying page
                var _CurrentPageSprite   :Sprite;

                // getter function to get the number of pages
                public function get numPages():int { return _PageList.length }

                public function Player():void {
                        // initialize the page list array
                        _PageList = new Array();
                        // set the index to 0 - the first page
                        _CurrentPageIdx = 0;
                        // set the page layer target
                        _PageDisplayTarget = pagelayer_mc;

                        // add events to the navigation buttons
                        next_btn.addEventListener(MouseEvent.CLICK, onNextClick);
                        prev_btn.addEventListener(MouseEvent.CLICK, onPrevClick);
                        exit_btn.addEventListener(MouseEvent.CLICK, onExitClick);

                        // setup the pages
                        addNewPage("Course Introduction", "page1");
                        addNewPage("The Content", "page2");
                        addNewPage("Conclusion", "page3");

                        trace("Ready to go ...");

                        // show the current page
                        displayCurrentPage();
                }

                // show the current page in the interface
                private function displayCurrentPage():void {
                        // is there a page currently showing? remove it
                        if (_CurrentPageSprite) _PageDisplayTarget.removeChild(_CurrentPageSprite);
                        // get a reference to the Sprite in the library
                        var _ClassRef:Class = getDefinitionByName(_PageList[_CurrentPageIdx].linkage) as Class;
                        _CurrentPageSprite =  new _ClassRef() as Sprite;
                        // show the page - add it to the display list of the page layer
                        _PageDisplayTarget.addChild(_CurrentPageSprite);
                        // update the interface based on the current page
                        updateInterface();
                }
```

```
		// updates the text fields and buttons based on the current page
		private function updateInterface():void {
			// have to add 1 to the current page index variable since array index start at 0
			pagenumber_txt.text = "Page " + (_CurrentPageIdx + 1) + " of " + numPages;
			// update the bread crumb with the page title
			breadcrumb_txt.text = _PageList[_CurrentPageIdx].title;
			// turn the navigation on or off based on where the learner is
			if (isNextPage()) next_btn.enabled = true;
				else next_btn.enabled = false;
			if (isPreviousPage()) prev_btn.enabled = true;
				else prev_btn.enabled = false;
		}

		// is there a next page from the current page?
		private function isNextPage():Boolean {
			// if the current page index greater than or the total number of pages?
			if (_CurrentPageIdx >= numPages - 1) return false;
			return true;
		}

		// is there a previous page from the current page?
		private function isPreviousPage():Boolean {
			// is the learner on the first page?
			if (_CurrentPageIdx == 0) return false;
			return true;
		}


		// do this when the next button is clicked
		private function onNextClick(e:Event):void {
			// is there a next page? if so increment the index and show the new page
			if (isNextPage()) {
				_CurrentPageIdx++
				displayCurrentPage()
			}
		}

		// do this when the previous button is clicked
		private function onPrevClick(e:Event):void {
			// is there a previous page? if so decrement the index and show the new page
			if (isPreviousPage()) {
				_CurrentPageIdx--
				displayCurrentPage()
			}
		}

		// do this when the exit button is clicked
		private function onExitClick(e:Event):void {
			// the learner should be presented with a confirmation dialog box here,
			// but it's beyond the scope of this demo
		}

		// will add a new page object to the page list
		// title is the name of the page, linkage is the ID of the page sprite in the library
		private function addNewPage(title:String, linkage:String):void {
			// create a new object to hold details of the page
			var pObject:Object = new Object();
			// assign the title and linkage to properties of the object
			pObject.title = title;
			pObject.linkage = linkage;
			// add it to the page list
			_PageList.push(pObject);
			trace("Added page '"+title+"' to the course");
		}
	}
}
```

Step 8 – Establishing LMS communication

Adding LMS tracking functionality is pretty easy if you've kept up to this point. All that's needed is to create an instance of the LMSData class and add a few lines of code. The LMSData class is one that I wrote to wrap the Pipwerks class and make it even easier to use since it handles initialization and removes the need to memorize the cmi variables that SCORM uses.

The first thing that we need to do is initialize the connection to the LMS. If the connection can't be created, then the initialize function will return a value of false and none of the calls to the LMSData instance will do anything. If it can create a connection, it gets the "last location," or course bookmark, and sets that to be our current page index. On each page change (our "displayCurrentPage()" function) it sets current page index to the LMS as the "last location." This function also calls a new "checkCourseCompleteion()" function which performs a simple completion test – is the learner on the last page? If so, then the course is marked complete. When the exit button is clicked, the "exitCourse()" function is called which closes the LMS connection and closes the browser window.

```
package {

        // import the classes so that we can work with the items on the screen
        import flash.display.*;
        import flash.text.*;
        // so that it can respond to events
        import flash.events.*;
        // utility class that let's us get references to the page Sprites in the library
        import flash.utils.getDefinitionByName;

        // a document class must expent either Sprite or Movieclip
        // since all of our code is in the document class and we have no timeline animation, we'll extend the Sprite class
        public class  Player extends Sprite      {

                // create an array to hold the information for the pages
                var _PageList                   :Array;
                // index of the current page
                var _CurrentPageIdx:     int;
                // reference to the "page layer" in the user interface
                var _PageDisplayTarget   :Sprite;
                // the sprite of the currently displaying page
                var _CurrentPageSprite   :Sprite;

                // an instance of the LMSData class to communicate with the LMS
                var _LMSData                    :LMSData;

                // getter function to get the number of pages
                public function get numPages():int { return _PageList.length }

                public function Player():void {
                        // initialize the page list array
                        _PageList = new Array();
                        // set the index to 0 - the first page
                        _CurrentPageIdx = 0;
                        // set the page layer target
                        _PageDisplayTarget = pagelayer_mc;
                        // instantiate the LMSData class
                        _LMSData = new LMSData();

                        // initialize the LMS connection
                        if (_LMSData.initialize()) {
                                trace("Successfully connected to the LMS!");
                                // set the current page index to the bookmark, but validate it first
                                var lastLoc:int = int(_LMSData.lastLocation);
                                if(lastLoc > 0) _CurrentPageIdx = lastLoc;
                        } else {
                                trace("There was a problem connecting to the LMS!");
                        }

                        // add events to the navigation buttons
```

```
			next_btn.addEventListener(MouseEvent.CLICK, onNextClick);
			prev_btn.addEventListener(MouseEvent.CLICK, onPrevClick);
			exit_btn.addEventListener(MouseEvent.CLICK, onExitClick);

			// setup the pages
			addNewPage("Course Introduction", "page1");
			addNewPage("The Content", "page2");
			addNewPage("Conclusion", "page3");

			trace("Ready to go ...");

			// show the current page
			displayCurrentPage();
		}

		// show the current page in the interface
		private function displayCurrentPage():void {
			// is there a page currently showing? remove it
			if (_CurrentPageSprite) _PageDisplayTarget.removeChild(_CurrentPageSprite);
			// get a reference to the Sprite in the library
			var _ClassRef:Class = getDefinitionByName(_PageList[_CurrentPageIdx].linkage) as Class;
			_CurrentPageSprite =  new _ClassRef() as Sprite;
			// show the page - add it to the display list of the page layer
			_PageDisplayTarget.addChild(_CurrentPageSprite);
			// update the interface based on the current page
			updateInterface();

			// tell the LMS what page the learner is on
			// just use the current page index as a simple bookmark
			_LMSData.lastLocation = String(_CurrentPageIdx);
			// check the course completion status
			checkCourseCompletion();
		}

		// updates the text fields and buttons based on the current page
		private function updateInterface():void {
			// have to add 1 to the current page index variable since array index start at 0
			pagenumber_txt.text = "Page " + (_CurrentPageIdx + 1) + " of " + numPages;
			// update the bread crumb with the page title
			breadcrumb_txt.text = _PageList[_CurrentPageIdx].title;
			// turn the navigation on or off based on where the learner is
			if (isNextPage()) next_btn.enabled = true;
				else next_btn.enabled = false;
			if (isPreviousPage()) prev_btn.enabled = true;
				else prev_btn.enabled = false;
		}

		// check the course completion status
		private function checkCourseCompletion():void {
			// simple completion check - if on the last page, then complete
			if (_CurrentPageIdx >= numPages - 1) {
				trace("COURSE COMPLETED!");
				_LMSData.setComplete();
			}
		}

		// is there a next page from the current page?
		private function isNextPage():Boolean {
			// if the current page index greater than or the total number of pages?
			if (_CurrentPageIdx >= numPages - 1) return false;
			return true;
		}

		// is there a previous page from the current page?
		private function isPreviousPage():Boolean {
			// is the learner on the first page?
			if (_CurrentPageIdx == 0) return false;
			return true;
		}
```

```
		// do this when the next button is clicked
		private function onNextClick(e:Event):void {
			// is there a next page? if so increment the index and show the new page
			if (isNextPage()) {
				_CurrentPageIdx++
				displayCurrentPage()
			}
		}

		// do this when the previous button is clicked
		private function onPrevClick(e:Event):void {
			// is there a previous page? if so decrement the index and show the new page
			if (isPreviousPage()) {
				_CurrentPageIdx--
				displayCurrentPage()
			}
		}

		// do this when the exit button is clicked
		private function onExitClick(e:Event):void {
			// the learner should be presented with a confirmation dialog box here,
			// but it's beyond the scope of this demo
			// exit the course
			_LMSData.exitCourse();
		}

		// will add a new page object to the page list
		// title is the name of the page, linkage is the ID of the page sprite in the library
		private function addNewPage(title:String, linkage:String):void {
			// create a new object to hold details of the page
			var pObject:Object = new Object();
			// assign the title and linkage to properties of the object
			pObject.title = title;
			pObject.linkage = linkage;
			// add it to the page list
			_PageList.push(pObject);
			trace("Added page '"+title+"' to the course");
		}
	}
}
```

Next steps

Now that we've got a working framework, there are several more things that we could do to it:

- Add the ability to group pages in to lessons
- Creating a menu navigation structure
- Add quizzes
- Externalize the pages so that they each page is a SWF
- An XML based course structure file
- Change the code to use a Model-View-Controller design pattern
- Use events to handle page changing and updating
- And a whole lot more …

For more information, please contact:

Matt Perkins
kheavy@nudoru.com